AFRL-VA-WP-TP-2005-303

VARIABLE STEP-SIZE SELECTION METHODS FOR IMPLICIT INTEGRATION SCHEMES

David B. Doman
Raymond Holsapple
Ram V. Iyer

OCTOBER 2005

Approved for public release; distribution is unlimited.

STINFO Final Report

AIR VEHICLES DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542

NOTICE

This report was cleared for public release by the Air Force Research Laboratory Wright Site (AFRL/WS) Public Affairs Office (PAO) and is releasable to the National Technical Information Service (NTIS). It will be available to the general public, including foreign nationals.

PAO Case Number: AFRL/WS-05-2521 1 Nov 2005.

THIS TECHNICAL REPORT IS APPROVED FOR PUBLICATION.

/s/

David B. Doman
Senior Aerospace Engineer
Control Design and Analysis Branch
Air Force Research Laboratory
Air Vehicles Directorate

/s /

Deborah Grismer
Chief
Control Design and Analysis Branch
Air Force Research Laboratory
Air Vehicles Directorate

/s/

Brian W. Van Vliet
Chief
Control Sciences Division
Air Force Research Laboratory
Air Vehicles Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
October 2005	Journal Article Preprint	10/01/2004 – 10/01/2005

4. TITLE AND SUBTITLE
VARIABLE STEP-SIZE SELECTION METHODS FOR IMPLICIT INTEGRATION SCHEMES

5a. CONTRACT NUMBER
In-house

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
N/A

6. AUTHOR(S)
David B. Doman (AFRL/VACA)
Raymond Holsapple and Ram V. Iyer

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Control Design and Analysis Branch (AFRL/VACA)
Control Sciences Division
Air Vehicles Directorate
Air Force Research Laboratory, Air Force Materiel Command
Wright-Patterson AFB, OH 45433-7542

Texas Tech University
Dept. of Mathematics
and Statistics
Lubbock, TX 79409

8. PERFORMING ORGANIZATION REPORT NUMBER
AFRL-VA-WP-TP-2005-303

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Vehicles Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson Air Force Base, OH 45433-7542

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/VACA

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-VA-WP-TP-2005-303

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Preprint submitted to *Journal of Computational Physics*. This technical paper contains color.

14. ABSTRACT
Implicit integration schemes are widely used in mathematics and engineering to solved ordinary differential equations. Every integration method requires one to choose a step-size for the independent variable which affects the efficiency and accuracy of the scheme. As every implicit integration scheme has a global error inherent to the scheme, we choose the total number of computations in order to achieve a prescribed global error as a measure of efficiency. A systematic method for choosing step-size is presented which is based on minimizing an efficiency function. The approach is applied to two popular numerical integration schemes.

15. SUBJECT TERMS
Numerical integration, Numerical analysis, Simulation, Ordinary differential equations, Runge-Kutta, Runge-Kutta-Nyström, numerical integration, variable step-size, implicit integration schemes

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. David B. Doman
Unclassified	Unclassified	Unclassified	SAR	50	19b. TELEPHONE NUMBER *(Include Area Code)* (937) 255-8451

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

Variable Step-Size Selection Methods for Implicit Integration Schemes

Raymond Holsapple, Ram Iyer

Dept. of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409

David Doman

USAF Research Laboratory, Wright-Patterson Air Force Base, OH 45433

Abstract

Implicit integration schemes, such as Runge-Kutta and Runge-Kutta-Nyström methods, are widely used in mathematics and engineering to numerically solve ordinary differential equations. Every integration method requires one to choose a step-size, h, for the integration. If h is too large or too small the efficiency of an implicit scheme is relatively low. As every implicit integration scheme has a global error inherent to the scheme, we choose the total number of computations in order to achieve a prescribed global error as a measure of efficiency of the integration scheme. In this paper, we propose the idea of choosing h by minimizing an efficiency function for general Runge-Kutta and Runge-Kutta-Nyström integration routines. This efficiency function is the critical component in making these methods variable step-size methods. We also investigate solving the intermediate stage values of these routines using both Newton's method and Picard iteration. We then show the efficacy of this approach on some standard problems found in the literature.

Key words: Runge-Kutta, Runge-Kutta-Nyström, numerical integration, variable

1 Introduction

Recently, there has been interest in the literature concerning the use of geometric integration methods, which are numerical methods that preserve some geometric quantities. For example, the symplectic area of a Hamiltonian system is one such concern in recent literature [1–4]. Tan [5] explores this concept using implicit Runge-Kutta integrators. Hamiltonian systems are of particular interest in applied mathematics, and in fact we test our variable step-size selection method on a well-known Hamiltonian system in Section 4.2. Furthermore, Hairer and Wanner [6,7] showed that although implicit Runge-Kutta methods can be difficult to implement, they possess the strongest stability properties. These properties include A-stability and A-contractivity (algebraic stability). These are the main reasons we choose to investigate variable integration step-size selection using Runge-Kutta methods.

First order ordinary differential equations are solved numerically using many different integration routines. Among the most popular methods are Runge-Kutta methods, multistep methods and extrapolation methods. Hull, Enright, Fellen and Sedgwick [8] have written an excellent comparison of these types of methods. They test a number of Runge-Kutta methods against multistep methods based on Adams formulas and an extrapolation method due to Bulirsch and Stoer [9]. A goal of that paper was to compare these different types

Email addresses: raymond.w.holsapple@ttu.edu (Raymond Holsapple), ram.iyer@ttu.edu (Ram Iyer), david.doman@wpafb.af.mil (David Doman).

of methods as to how they handle routine integration steps under a variety of accuracy requirements. Implicit or explicit integration methods require one to choose a step-size, h, for the integration. One of the questions Bulirsch and Stoer investigate is a strategy for deciding what step-size h to use as the methods progress from one step to another. Others have investigated this very same problem in the past [8,10–12].

In this paper, we propose the idea of choosing variable step-sizes by minimizing an efficiency function for general Runge-Kutta and Runge-Kutta-Nyström integration routines. As every implicit integration scheme has a global error inherent to the scheme, we choose the total number of computations in order to achieve a prescribed global error as a measure of efficiency of the integration scheme. For illustration purposes, consider Figure 1.0.1, referring to the solution of (2). Let $\tilde{x}(t_k)$ be our approximation to $x(t_k)$. We determine the variable step-sizes $h_1, h_2, \ldots, h_8$, where $h_k = t_k - t_{k-1}$, so that we minimize an efficiency function that minimizes the sum of the total number of computations to compute $\tilde{x}(t_k)$ for $k = 1, 2, \ldots, 8$ and the global error that propagates from the local truncation errors at each step of integration. An alternate and perhaps simple way of understanding our method is that we choose $h\overline{L}\|A\|$, where $\overline{L}$ is a Lipschitz constant for the differential equation and A is a matrix that describes the integration scheme used to integrate the differential equation. Of course, in a fixed step-size method one chooses h. To the best of our knowledge, our proposed method is novel.

In the rest of this section, we briefly describe the approaches found in the literature. Hull, Enright, Fellen and Sedgwick [8] approach this topic as follows. First, they determine $h_{\max}$, which is a measure of the "scale" of a problem. This helps to allow them from not stepping past any interesting fluctuations in the

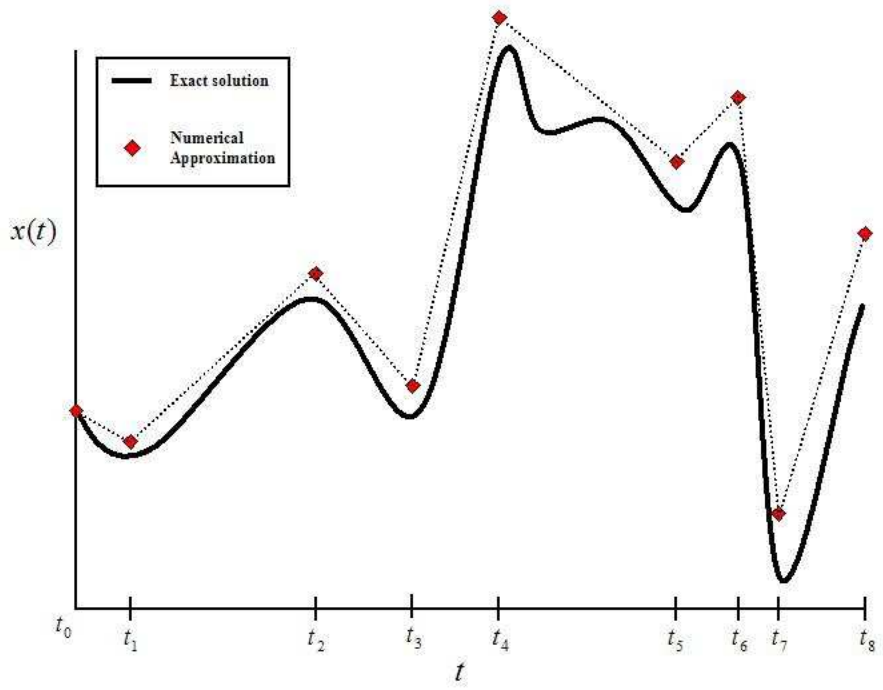

Fig. 1.0.1. Illustration of variable step-sizes and error propagation in numerical integration

solution. Then, for their Runge-Kutta methods they compute τ, an estimate on the local truncation error, which must be bounded by the tolerance, ε. They then compute

$$h_{\text{new}} = \min\left\{h_{\max}, 0.9h_{\text{old}}\left(\varepsilon/\tau\right)^{1/p}\right\}. \tag{1}$$

where p is the order of the Runge-Kutta routine being used.

Stoer and Bulirsch [10] arrive at a very similar solution to this problem. To describe what they do, we first note that throughout this paper, we will consider solving the following first order ordinary differential equation:

$$\frac{dx}{dt} = f(t,x), \quad x(0) = x_0 \in \mathbb{R}^n, \tag{2}$$

where $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz continuous in the second argument, i.e.

for any $t \in \mathbb{R}$ and any vectors $x \in \mathbb{R}^n, y \in \mathbb{R}^n$, we have

$$\|f(t,x) - f(t,y)\| \leq L(t)\|x - y\|, \tag{3}$$

where $L(\cdot) \in L^\infty[0,T]$. Stoer and Bulirsch consider two discretization methods, Φ_1 and Φ_2, of Runge-Kutta type to solve (2), the first with order p and the second with order $p+1$, i.e. we have

$$\hat{x}_{k+1} = \bar{x}_k + h_{\text{old}}\Phi_1(t_k, \bar{x}_k; h_{\text{old}}) \tag{4}$$
$$\bar{x}_{k+1} = \bar{x}_k + h_{\text{old}}\Phi_2(t_k, \bar{x}_k; h_{\text{old}}) \tag{5}$$

Denoting the tolerance by ε, and given a current step-size, h_{old}, they obtain:

$$h_{\text{new}} = h_{\text{old}} \left| \frac{\varepsilon}{\bar{x}_{k+1} - \hat{x}_{k+1}} \right|^{1/(p+1)}. \tag{6}$$

Stoer and Bulirsch go on to recommend after extensive numerical experimentation, that equation (6) be altered to

$$h_{\text{new}} = 0.9h_{\text{old}} \left| \frac{\varepsilon h_{\text{old}}}{\bar{x}_{k+1} - \hat{x}_{k+1}} \right|^{1/p}. \tag{7}$$

As one can see, formulas (1),(6) and (7) depend on some measure of the local error at the $(k+1)$st step of integration. Stoer and Bulirsch [10] also point out that there is another way to determine h_{new}, but it requires one to estimate higher order derivatives of f. For example, a fourth order Runge-Kutta method would require one to estimate derivatives of f of the fourth order. Not only is this very costly, but this other method uses the local truncation error at the kth step of integration.

Houwen [11] took a similar approach to adjusting step-sizes. Again let ε be the tolerance. Houwen then forms a discrepance function $d(t_k, x_k; h_{\text{old}})$ at the point (t_k, x_k). Then the new step-size is determined to be the solution of the

equation

$$\|d(t_k, x_k; h_{\text{old}})\| = \varepsilon. \tag{8}$$

Houwen considers three types of discrepance functions:

(1) an approximation to the local discretization error

(2) the residual term left when the local difference solution is submitted into the differential equation

(3) the discrepance of linearity of the differential equation

The first two clearly are functions that are some measure of local error of the difference scheme being used. The discrepance of linearity method is merely a way to choose h_{new} such that the Jacobian matrix for non-linear systems does not change very much,(i.e. within some tolerance ε), over the interval $[t_k, t_k + h_{\text{new}}]$. This method also deals with some measure of local stability of the differential equation.

Cano and Duran [12] investigate variable step-size selection using linear multistep methods. Again consider equation (2). Given a tolerance ε, they let

$$h_n = \varepsilon s(x(t_n), \varepsilon) + \mathcal{O}(\varepsilon^p), \tag{9}$$

where p is the order of the method and s is function satisfying the following:

(1) $s_{\min} \leq s(x, \varepsilon) \leq s_{\max}$, with $s_{\min}, s_{\max} > 0$,

(2) s is C^∞ in both arguments and all the derivatives of s are bounded.

2 Implicit Integration Methods

Numerical methods for solving initial value problems such as (2) may be either explicit or implicit. The focus of this paper is concentrated on using implicit methods. In this section, we describe two classes of implicit numerical integration schemes and how one might use the methods to solve (2). We assume the solution exists for $t \in [0, T]$, with $T > 0$.

2.1 Runge-Kutta Methods

A detailed description of a general s-stage Runge-Kutta method for the solution of (2) can be found in many publications [1,3,13]. The general method for a fixed step-size, h, is described below:

$$y_{i_k} = x_k + h \sum_{j=1}^{s} a_{ij} f(t_k + c_j h, y_{j_k}), \quad i = 1, ..., s, \tag{10}$$

$$x_{k+1} = x_k + h \sum_{i=1}^{s} b_i f(t_k + c_i h, y_{i_k}), \quad x_0 = x(0). \tag{11}$$

In the above equations, the y_{i_k} are stage-values that must be computed at every step of the integration, and x_k approximates the exact solution $x(t_k)$ at the point $t_k = kh$, where h is the fixed step-size of integration. The a_{ij} and b_i are unique to any particular Runge-Kutta scheme and the c_i satisfy

$$c_i = \sum_{j=1}^{s} a_{ij} \ , \quad i = 1, \ldots, s \tag{12}$$

For notational purposes, define the following:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & \cdots & a_{ss} \end{bmatrix}, \; Y_k = \begin{bmatrix} y_{1_k} \\ y_{2_k} \\ \vdots \\ y_{s_k} \end{bmatrix}, \; c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_s \end{bmatrix}, \; X_k = \begin{bmatrix} x_k \\ x_k \\ \vdots \\ x_k \end{bmatrix}, \; \bar{A} = A \otimes I \tag{13}$$

where I is the $n \times n$ identity matrix and $\otimes$ is the Kronecker product. In other words, $\bar{A}$ is the $ns \times ns$ matrix direct product of A and I. Furthermore, consider the function $\tilde{f} : \mathbb{R} \times \mathbb{R}^{ns} \to \mathbb{R}^{ns}$ defined by

$$\tilde{f}(t_k, Y_k) = \begin{bmatrix} f(t_k + c_1 h, y_{1_k}) \\ f(t_k + c_2 h, y_{2_k}) \\ \vdots \\ f(t_k + c_s h, y_{s_k}) \end{bmatrix}. \tag{14}$$

Now we can write the system of ns equations given in equation (10) as

$$Y_k = X_k + h\bar{A}\tilde{f}(t_k, Y_k). \tag{15}$$

For each k this is an implicit equation involving the vectors $\{y_{i_k}\}_{i=1}^{s}$. Equation (15) can be solved using Newton's method or fixed point iteration (Picard iteration). Let's consider Picard iteration. We solve (15) for each k using the following iterative scheme:

$$Y_k^{j+1} = X_k + h\bar{A}\tilde{f}\left(t_k, Y_k^j\right) = F\left(t_k, Y_k^j\right). \tag{16}$$

For any fixed k, the iterative scheme given in (16) will converge to the solution

of (15) provided that F satisfies a favorable condition. The following theorem addresses this convergence.

Theorem 2.1 *Consider the iterative scheme given by (16). Let $L(t)$ be the function from (3), and let A be the $s \times s$ matrix from (13). If $hL(t_k)\|A\| < 1$ then there exists a unique vector $\overline{Y} \in \mathbb{R}^{ns}$ such that $F(t_k, \overline{Y}) = \overline{Y}$ for any point $t_k \in [0, T]$ that is fixed. Furthermore, the sequence $Y_k^{j+1} = F(t_k, Y_k^j)$ converges linearly to $\overline{Y}$.*

Proof. First, let us refer to a very useful result concerning the norms of Kronecker products. According to Van Loan [14], we get

$$\left\|\bar{A}\right\| = \|A \otimes I\| = \|A\| \cdot \|I\| = \|A\|. \tag{17}$$

Next, we show that F is Lipschitz continuous in the second argument with Lipschitz constant $hL(t_k)\|A\|$. We will begin by finding the Lipschitz constant of the function $\tilde{f}$. Choose any $t_k \in [0, T]$ and any vectors u and v in $\mathbb{R}^{ns}$ where $u = [(u_1)^T \;\; (u_2)^T \;\; \cdots \;\; (u_s)^T]^T$ and $u_i \in \mathbb{R}^n$ for $i = 1, \ldots, s$. Similarly form the vector v.

$$\begin{aligned}
\left\|\tilde{f}(t_k, u) - \tilde{f}(t_k, v)\right\| &= \left\|\left[(f(t_k, u_1) - f(t_k, v_1))^T \;\; \cdots \;\; (f(t_k, u_s) - f(t_k, v_s))^T\right]^T\right\| \\
&\leq L(t_k)\left\|\left[(u_1 - v_1)^T \;\; \cdots \;\; (u_s - v_s)^T\right]^T\right\| && (18) \\
&= L(t_k)\left\|\left[(u_1)^T \;\; \cdots \;\; (u_s)^T\right]^T - \left[(v_1)^T \;\; \cdots \;\; (v_s)^T\right]^T\right\| && (19) \\
&= L(t_k)\,\|u - v\| && (20)
\end{aligned}$$

Now we may compute the Lipschitz constant of F. Again, choose any $t_k \in [0, T]$ and any vectors u, $v \in \mathbb{R}^{ns}$.

$$\|F(t_k,u)-F(t_k,v)\| = \left\|X_k + h\bar{A}\tilde{f}(t_k,u) - X_k - h\bar{A}\tilde{f}(t_k,v)\right\| \tag{21}$$
$$= h\left\|\bar{A}\left(\tilde{f}(t_k,u)-\tilde{f}(t_k,v)\right)\right\| \tag{22}$$
$$\leq h\left\|\bar{A}\right\|\left\|\tilde{f}(t_k,u)-\tilde{f}(t_k,v)\right\| \tag{23}$$
$$\leq hL(t_k)\left\|\bar{A}\right\|\|u-v\| \tag{24}$$
$$= hL(t_k)\|A\|\|u-v\|. \tag{25}$$

This shows that F is Lipschitz continuous in the second argument with Lipschitz constant $hL(t_k)\|A\|$. Since $hL(t_k)\|A\| < 1$, we may apply the Contractive Mapping Theorem to F, [15]. Hence, there exists a unique point $\overline{Y} \in \mathbb{R}^{ns}$ such that $F(t_k, \overline{Y}) = \overline{Y}$ for any point $t_k \in \mathbb{R}$ that is fixed. The theorem also ensures that $\overline{Y}$ must be the limit of every sequence obtained from (16) with a starting point $Y_k^0 \in \mathbb{R}^{ns}$. □

Theorem 2.1 suggests how one might implement equations (11) and (16) to solve (2) on $[0,T]$. The starting vector $x_0 \in \mathbb{R}^n$ is known. In general, assume x_k is known. Use the following procedure to compute x_{k+1}.

(1) Choose a tolerance $\varepsilon > 0$ as small as you wish.

(2) Choose a starting guess for the s stage-values, and denote this guess as Y_k^0.

(3) For $j = 0, 1, 2, ...$, compute the following:
   (a) $Y_k^{j+1} = F\left(t_k, Y_k^j\right)$
   (b) $\delta = \left\|Y_k^{j+1} - Y_k^j\right\|$.

(4) If $\delta \leq \varepsilon$, let $Y_k = Y_k^{j+1}$.

(5) Use the s $n \times 1$ stage-value vectors determined in step four to explicitly compute x_{k+1}.

The method described above is known as Picard iteration; Newton's method might also be used to solve for the stage-values. A theorem on the convergence

of Newton's method is more complicated than Theorem 2.1; it is not sufficient to assume $hL(t_k)\|A\| < 1$ in order to guarantee that Newton's method converges. The Newton-Kantorovich Theorem [10,16,17] provides sufficient conditions for existence of a solution to the iteration and the uniqueness of that solution. To solve for the stage-values using Newton's method, step three should be replaced by the following:

(3) Define $G(t_k, Y_k) = F(t_k, Y_k) - Y_k$, and for $j = 0, 1, 2, ...$, compute the following:
   (a) $Y_k^{j+1} = Y_k^j - \left(DG\left(t_k, Y_k^j\right)^{-1}\right) G\left(t_k, Y_k^j\right)$
   (b) $\delta = \left\|G\left(t_k, Y_k^{j+1}\right)\right\|$

where DG represents the Jacobian matrix of G.

2.2 Runge-Kutta-Nyström Methods

Runge-Kutta-Nyström (RKN) methods are similar to Runge-Kutta methods, but are designed to solve second-order systems. Consider the following system:

$$\frac{d^2x}{dt^2} = f(t, x), \quad x(0) = x_0 \in \mathbb{R}^n, \quad \dot{x}(0) = v_0 \in \mathbb{R}^n, \tag{26}$$

which can be written as

$$\frac{dv}{dt} = f(t, x); \quad \frac{dx}{dt} = v, \quad x(0) = x_0, \quad v(0) = v_0. \tag{27}$$

We assume a solution exists on $[0, T]$ for $T > 0$. A general s-stage RKN method may be used to solve (27) on $[0, T]$, and is described by J.M. Sanz-Serna and M.P. Calvo [3] as follows:

$$y_{i_k} = x_k + h\gamma_i v_k + h^2 \sum_{j=1}^{s} a_{ij} f(t_k + \gamma_j h, y_{j_k}), \quad i = 1, ..., s \tag{28}$$

$$v_{k+1} = v_k + h \sum_{i=1}^{s} b_i f(t_k + \gamma_i h, y_{i_k}) \tag{29}$$

$$x_{k+1} = x_k + h v_k + h^2 \sum_{i=1}^{s} \beta_i f(t_k + \gamma_i h, y_{i_k}). \tag{30}$$

Exactly as with the Runge-Kutta methods, the y_{i_k} are the stage-values and must be solved for implicitly. In addition to the definitions in (13), define

$$\Gamma = \begin{bmatrix} \gamma_1 & 0 & \cdots & 0 \\ 0 & \gamma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \gamma_s \end{bmatrix}, V_k = \begin{bmatrix} v_k \\ v_k \\ \vdots \\ v_k \end{bmatrix}, \overline{\Gamma} = \Gamma \otimes I, \tag{31}$$

where I is the $n \times n$ identity matrix. Now, the ns equations in (28) may be written as

$$Y_k = X_k + h\overline{\Gamma} V_k + h^2 \bar{A} \tilde{f}(t_k, Y_k). \tag{32}$$

Again, we have an implicit equation for the s stage-value vectors $\{y_{i_k}\}_{i=1}^{s} \subset \mathbb{R}^n$.

Just as in Section 2.1, we may solve (32) using Newton's method or by using Picard iteration. If we use Picard iteration, we have the following iterative scheme:

$$Y_k^{j+1} = X_k + h\overline{\Gamma} V_k + h^2 \bar{A} \tilde{f}(t_k, Y_k^j) = H(t_k, Y_k^j). \tag{33}$$

Similar to the proof of Theorem 2.1, one can easily prove the following theorem.

Theorem 2.2 *Consider the iterative scheme given by (33). Let $L(t)$ be the function from (3), and let A be the $s \times s$ matrix from (13). If $h^2 L(t_k) \|A\| < 1$*

then there exists a unique vector $\overline{Y} \in \mathbb{R}^{ns}$ *such that* $H(t_k, \overline{Y}) = \overline{Y}$ *for any point* $t_k \in [0, T]$ *that is fixed. Furthermore, the sequence* $Y_k^{j+1} = H(t_k, Y_k^j)$ *converges linearly to* $\overline{Y}$*.*

Proof. See the proof of Theorem 2.1. □

If we choose to use Newton's method to solve (32) for the stage-values, it is not sufficient to assume $h^2 L(t_k)\|A\| < 1$. Once again, we may refer to Stoer and Bulirsch [10] for conditions that guarantee convergence of Newton's method.

3 Step-Size Selection

When implementing a Runge-Kutta (or Runge-Kutta-Nyström) numerical integration routine, we have shown it is sufficient to assume that $hL(t_k)\|A\| < 1$ (or $h^2 L(t_k)\|A\| < 1$) to guarantee convergence of the implicit scheme when using a Picard iteration. One might wonder though, is there an optimal choice, in the sense of computational efficiency, for h? If so, how might it be found?

3.1 Optimization Using Picard Iteration

Consider solving (2) numerically on the interval $[0, T]$ which is partitioned by the following sequence of points: $\{kh\}_{k=0}^{K}$. In the k-th sub-interval the convergence of the Picard iteration is linear, so the number of computations required for convergence, to within ε, to the fixed point of (16) can be written as a function of the Lipschitz constant of the function F: $N_k = \phi\left(hL(t_k)\|A\|\right)$. Then the total number of computations over the interval $[0, T]$ can be written as

$N(h) = \sum_{k=1}^{K} N_k$. In the following, we find an explicit expression for $\phi(\cdot)$ for Runge-Kutta methods. Consider the following inequalities:

$$\begin{aligned}
\left\|Y_k^{j+1} - Y_k^j\right\| &= \left\|F(t_k, Y_k^j) - F(t_k, Y_k^{j-1})\right\| && (34)\\
&\leq hL(t_k)\|A\| \left\|Y_k^j - Y_k^{j-1}\right\| && (35)\\
&\leq (hL(t_k)\|A\|)^2 \left\|Y_k^{j-1} - Y_k^{j-2}\right\| && (36)\\
&\vdots \\
&\leq (hL(t_k)\|A\|)^j \left\|Y_k^1 - Y_k^0\right\| && (37)\\
&= C_k \left(hL(t_k)\|A\|\right)^j, && (38)
\end{aligned}$$

where $C_k = \|Y_k^1 - Y_k^0\|$ is fixed for each k and depends on the guess Y_k^0. Since $hL(t_k)\|A\| < 1$, we must have $\left\|Y_k^{j+1} - Y_k^j\right\| \to 0$ as $j \to \infty$. Suppose we want $\delta = \left\|Y_k^{j+1} - Y_k^j\right\| \leq \varepsilon$; then, it is sufficient to have $C_k \left(hL(t_k)\|A\|\right)^j \leq \varepsilon$. As a stopping criterion in the k-th step of integration, we choose to stop the fixed point iteration at the smallest natural number N_k greater than or equal to M where M satisfies $C_k \left(hL(t_k)\|A\|\right)^M = \varepsilon$. Then we have

$$\begin{aligned}
M &= \frac{\ln\left(\varepsilon / C_k\right)}{\ln\left(hL(t_k)\|A\|\right)} && (39)\\
N_k &= \lceil M \rceil && (40)
\end{aligned}$$

Now let $C = \max_k C_k$ and $\overline{L} = \sup_{t\in[0,T]} L(t)$. In (39), ε and C_k depend on the user. Once these are chosen, the choice of h depends on the differential equation being solved, through the Lipschitz constant $L(t_k)$, and on the integration method being implemented, through $\|A\|$. We will try to minimize M by adjusting the choice of h to the problem parameters $L(t_k)$ and $\|A\|$. Notice that $C(h\overline{L}\|A\|)^M = \varepsilon$ implies that $C_k(hL(t_k)\|A\|)^M \leq \varepsilon$ for each k. Thus, we minimize the cost function

$$J_1(h) = K\frac{\ln(\varepsilon/C)}{\ln\left(h\overline{L}\|A\|\right)} \tag{41}$$

$$= \frac{T\ln(\varepsilon/C)}{h\ln\left(h\overline{L}\|A\|\right)}. \tag{42}$$

This is the same as maximizing the cost function

$$J_2(h) = \frac{h\ln\left(h\overline{L}\|A\|\right)}{T\ln(\varepsilon/C)}. \tag{43}$$

By computing $\arg\min J_2(h)$, one finds the step-size h that minimizes the number of computations for the iterative scheme to converge. If this were the only measure of optimality of concern, it is easily shown, through a calculus argument, that the cost function $J_2(h)$ is maximized when

$$h = \frac{1}{e\overline{L}\|A\|}. \tag{44}$$

However, one might also want the global error of the numerical solution to be as small as possible. Global error in any numerical integration scheme depends on the scheme being used. In this paper, we are concentrating on Runge-Kutta schemes. The global error for Runge-Kutta schemes also varies depending on the scheme one chooses to implement. For the purpose of explanation, let us consider the implicit midpoint rule. The implicit midpoint rule is a one-stage Runge-Kutta method where $a_{11} = \frac{1}{2}$ and $b_1 = 1$ in (10) and (11). The implicit midpoint method has global error $\mathcal{O}(Th^2)$. Then to find h, we alter the cost function J_2 and maximize the following cost function:

$$J_3(h) = \frac{h\ln(h\overline{L}\|A\|)}{T\ln(\varepsilon/C)} - \kappa T h^2, \tag{45}$$

where κ is a number to be chosen. First we compute

$$\frac{dJ_3}{dh} = \frac{\ln(h\overline{L}\|A\|) + 1}{T\ln(\varepsilon/C)} - 2\kappa T h \tag{46}$$

and

$$\frac{d^2 J_3}{dh^2} = \frac{1}{Th\ln(\varepsilon/C)} - 2\kappa T. \tag{47}$$

In order to find $\max_h J_3(h)$, we must find the $h > 0$ that solves

$$\frac{\ln(h\overline{L}\|A\|) + 1}{T\ln(\varepsilon/C)} - 2\kappa T h = 0 \tag{48}$$

such that

$$\frac{1}{Th\ln(\varepsilon/C)} - 2\kappa T < 0. \tag{49}$$

We must have T and h positive, and for a numerical solution to be meaningful, certainly ε must be very small and in general much less than C. Thus, $\ln(\varepsilon/C) < 0$. We then require $\kappa \geq 0$ to ensure that the second derivative of J_3 is negative, which guarantees that the solution to (48) is indeed a maximum.

Now let

$$\kappa = -\lambda^2 \frac{\overline{L}\|A\|}{2T^2\ln(\epsilon/C)} \ , \tag{50}$$

where λ is a free parameter that weights the optimization toward efficiency in time or toward global error. A better understanding of how λ affects the variable step-size selection process can best be explained by studying Table 4.1.2 and Table 4.2.2. By substituting this κ into (48), we find that we must solve

$$\ln(h\overline{L}\|A\|) + 1 + \lambda^2 h\overline{L}\|A\| = 0 \tag{51}$$

for h given an arbitrary value for λ. In practice, we actually make the substitution $x = h\overline{L}\|A\|$ and solve

$$\ln x + 1 + \lambda^2 x = 0 \tag{52}$$

for x. We then compute $h = \dfrac{x}{\overline{L}\|A\|}$. The solution to this equation exists and is unique. This is because the h that solves (51) is the unique global maximum of the function J_3, which exists because of the concavity of J_3. Furthermore,

(51) must be solved numerically for $\lambda \neq 0$; for example, Newton's method or Picard iteration may be used. For $\lambda = 0$, notice that the solution is $h = \dfrac{1}{e\overline{L}\|A\|}$ which was discussed earlier.

If one is interested in finding an equation similar to (51) for a Runge-Kutta method other than the implicit midpoint method, two things change in (45). First, $\|A\|$ will certainly change when the method changes. It will be necessary to change the second term as well. Suppose the method chosen has global error $\mathcal{O}(Th^r)$. Then, (45) becomes

$$\tilde{J}_3(h) = \frac{h \ln(h\overline{L}\|A\|)}{T \ln(\varepsilon/C)} - \kappa T h^r. \tag{53}$$

Now we define

$$\kappa = -\lambda^2 \frac{(\overline{L}\|A\|)^{r-1}}{2T^2 \ln(\varepsilon/C)}\ , \tag{54}$$

and discover that we must now solve

$$\ln x + 1 + \lambda^2 x^{r-1} = 0 \tag{55}$$

for x after again making the substitution $x = h\overline{L}\|A\|$.

3.2 Local Optimization Using Picard Iteration

If one considers the analysis presented in the previous section, an obvious question might arise. Is it possible to repeat the process while minimizing local error and local computations instead of global error and total computations? It turns out that the answer is no.

Instead of minimizing (41), i.e. maximizing (43), if we want to minimize the

total number of computations locally we minimize the cost function

$$\hat{J}_1(h) = \frac{\ln(\varepsilon/C)}{\ln\left(h\overline{L}\|A\|\right)}\,. \tag{56}$$

Equivalently, we can maximize

$$\hat{J}_2(h) = \frac{\ln\left(h\overline{L}\|A\|\right)}{\ln(\varepsilon/C)}\,. \tag{57}$$

We also want to consider the local truncation error. Just as in the previous section, we choose the implicit midpoint method for explanation purposes. The local truncation error for this method is $\mathcal{O}(h^3)$. Thus, similar to (45), we want to maximize the cost function

$$\hat{J}_3(h) = \frac{\ln\left(h\overline{L}\|A\|\right)}{\ln(\varepsilon/C)} - \kappa h^3\,. \tag{58}$$

First, we compute

$$\frac{d\hat{J}_3}{dh} = \frac{1}{h\ln(\varepsilon/C)} - 3\kappa h^2 \tag{59}$$

and set it equal to zero. After doing so and rearranging, we get

$$h^3 = \frac{1}{3\kappa\ln(\varepsilon/C)}. \tag{60}$$

Since $\varepsilon \ll C$ in general, (e.g. in our implementations we choose $\varepsilon = 10^{-10}$), it will certainly be the case that $\ln(\varepsilon/C) < 0$. Thus, (60) would imply that it is necessary to have $\kappa < 0$, since it must be that $h > 0$.

Next, we compute

$$\frac{d^2\hat{J}_3}{dh^2} = \frac{-1}{h^2\ln(\varepsilon/C)} - 6\kappa h. \tag{61}$$

Since we are trying to maximize (58), we need $\dfrac{d^2\hat{J}_3}{dh^2} < 0$. However, if we consider each term in (61), we can easily see that this is impossible. Since $\ln(\varepsilon/C) < 0$, it must be that $\dfrac{-1}{h^2\ln(\varepsilon/C)} > 0$. Since we have determined

that we must have $\kappa < 0$, it must also be that $-6\kappa h > 0$. Hence, the second derivative would imply that the function we are optimizing is actually concave up and has no local (or absolute) maximum for $h > 0$.

The analysis above shows that this method of choosing variable step-sizes (minimization of a particular efficiency function) fails if one considers local truncation error and local computations for the efficiency function. Although, there certainly are ways of choosing h based on local truncation errors as we described in the Introduction, we simply cannot use local error and computations when using the methods presented in this paper.

3.3 Lipschitz Constant Unknown

For most initial value problems the Lipschitz constant of f is not easily accessible. In this case, an approach that is slightly different than that of Section 3.1 is taken to optimize h. The idea in this case is to linearize the function f at each step of integration by computing the Jacobian of f. We essentially find an optimal h at each step of the integration using the analysis from Section 3.1. The method is described in detail below:

(1) Choose a value for the parameter λ. (A method for choosing λ will be given in Section 4.1.)
(2) Solve equation (52) for x once.
(3) At $t = 0$, let $\overline{L} = \|Df\|$, where Df is the Jacobian matrix of f, and compute $h = \dfrac{x}{\overline{L}\|A\|}$.
(4) Perform one step of integration using the implicit midpoint rule.
(5) Recompute $\overline{L}$ using the new values of the state variables, and use this $\overline{L}$

to find a new optimal h.

(6) Repeat steps four and five until the integration reaches $t = T$.

3.4 Optimization Using Newton's Method

We mentioned in Section 2.1 that one might also use Newton's method to solve for the stage values Y_k. Because of this, the analysis for finding an optimal value of h can be repeated for Newton's method. Convergence properties, and hence convergence theorems, are more complicated for Newton's method than for fixed point iteration. Because of this, one might expect the analysis to be a bit more involved than that of equations (34)-(51).

Before we begin looking at the optimization process for Newton's method, let us first consider the following Lemma.

Lemma 3.1 *If R is an invertible $n \times n$ matrix, then*

$$\left\|R^{-1}\right\| \leq \frac{\|R\|^{n-1}}{|\det R|}. \tag{62}$$

Proof. Let $\lambda_i\left(R^T R\right)$ denote the i-th eigenvalue of $R^T R$. Since $R^T R > 0$, we have $\lambda_i\left(R^T R\right) > 0$ for $i = 1, \ldots, n$ and $\left(R^T R\right)^{-1} > 0$. Now using a well-known property of matrices and the Rayleigh-Ritz inequality, we get the following:

$$\left\|R^{-1}\right\|^2 = \lambda_{\max}\left(\left(R^T R\right)^{-1}\right) \tag{63}$$

$$= \frac{1}{\lambda_{\min}\left(R^T R\right)} \tag{64}$$

$$= \frac{\prod_{i=1}^{n} \lambda_i\left(R^T R\right)}{\lambda_{\min}\left(R^T R\right) \cdot \det\left(R^T R\right)} \tag{65}$$

$$\leq \frac{\left[\lambda_{\max}\left(R^T R\right)\right]^{n-1}}{\left(\det R\right)^2} \tag{66}$$

$$= \frac{\|R\|^{2(n-1)}}{\left(\det R\right)^2} \tag{67}$$

$$= \left(\frac{\|R\|^{n-1}}{\det R}\right)^2 . \tag{68}$$

Taking square roots on both sides of the above inequality yields the desired result. □

Again, we consider solving (2) numerically on the interval $[0, T]$ which is partitioned by the following sequence of points: $\{kh\}_{k=0}^{K}$. Stoer and Bulirsch [10] prove a theorem showing that Newton's method is quadratically convergent. We will find an optimal choice of h using the results of that theorem. We begin by defining

$$\rho_k = \frac{\alpha_k \beta_k \gamma}{2} , \tag{69}$$

where α_k, β_k and γ are described below. First we let $\overline{C}$ be a convex subset of $\mathbb{R}^{ns}$ and mention that the theorem from [10] and the analysis below is valid only in a neighborhood, $S_r\left(Y_k^0\right)$, of the initial guess Y_k^0 such that $S_r\left(Y_k^0\right) \subset \overline{C}$. We then let

$$\alpha_k = \left\| Y_k^1 - Y_k^0 \right\| \ , \quad k = 1, \ldots, K \tag{70}$$

$$\beta_k = \sup_{Y \in \overline{C}} \frac{\| DG\left(t_{k-1}, Y\right) \|^{ns-1}}{\left| \det DG\left(t_{k-1}, Y\right) \right|} \ , \quad k = 1, \ldots, K \tag{71}$$

$$\gamma = h\overline{L}\|A\|, \tag{72}$$

where $\overline{L}$ is a constant that satisfies

$$\left\| D\tilde{f}(t,u) - D\tilde{f}(t,v) \right\| \leq \overline{L}\|u - v\|, \tag{73}$$

for any vectors u and v in $\mathbb{R}^{ns}$. Note that this analysis holds whether the Lipschitz constant of $D\tilde{f}$, $\overline{L}$, is given from the beginning or if it is approximated similarly to what was done in the previous section; we shall only require that $\overline{L}$ be a known constant at the k-th step of integration.

It should also be noted that in practice, i.e. in the example we show later, α_k and β_k will not depend on k. For implementation purposes, we elect to choose Y_k^0 to be the same vector for all k. Hence, α_k will be the same for each k. In general, this is not necessary; that is why we define α below for convenience of analysis. Actually, β_k must satisfy $\left\| DG\left(t_{k-1}, Y\right)^{-1} \right\| \leq \beta_k$ for all $Y \in \overline{C}$. We then apply Lemma 3.1 to arrive at the definition given by equation (71). Now, β_k depends on the step-size through the Jacobian matrix of G, i.e. through DG. Since this is a variable step-size method, this implies that β_k should be computed at each step of the integration using the current step-size. We quickly found that computing β_k at each step of the integration makes the process computationally inefficient. Instead, we approximate $\beta = \max_k \beta_k$ before solving a particular example by first solving the system on a much smaller time interval. As we solve the problem, we keep track of the current values of β_k and keep the largest value to use as a global constant to solve the entire example. Putting all this together and using the theorem from Stoer

and Bulirsch, we know that for k fixed, the iterations of Newton's method must satisfy

$$\left\|Y_k^{j+1} - Y_k^j\right\| \leq \alpha_k \rho_k^{2^j-1} \leq \alpha \rho_k^{2^j-1}, \tag{74}$$

where $\alpha = \max_k \alpha_k$.

Furthermore, we must assume (as do Stoer and Bulirsch) that $\rho_k < 1$. Then for each k, it is sufficient to stop iterating Newton's method at the smallest natural number N_k greater than or equal to M, where M satisfies $\alpha \rho_k^{2^M-1} = \varepsilon$. Solving for M, we get

$$M = \log_2 \left(\frac{\ln(\rho_k \varepsilon \alpha^{-1})}{\ln \rho} \right). \tag{75}$$

Again, we choose $N_k = \lceil M \rceil$. Also, we are showing the analysis for the implicit midpoint method which has global error $\mathcal{O}(Th^2)$. Thus, we minimize the following cost function:

$$J_4(\rho) = \frac{T\alpha\beta\overline{L}\|A\|}{2\rho_k} \log_2 \left(\frac{\ln(\rho_k \varepsilon \alpha^{-1})}{\ln \rho} \right) + \frac{4\kappa T \rho_k^2}{(\alpha\beta\overline{L}\|A\|)^2}. \tag{76}$$

We now define

$$\kappa = \lambda^2 \frac{(\alpha\beta\overline{L}\|A\|)^3}{16 \ln 2}. \tag{77}$$

Then, after using a little calculus similar to equations (45)-(49), we find the optimal h by first finding the ρ_k that solves

$$\left(\ln(\rho_k \varepsilon \alpha^{-1})\right)^{-1} - (\ln \rho_k)^{-1} - (\ln 2) \log_2 \left(\frac{\ln(\rho_k \varepsilon \alpha^{-1})}{\ln \rho_k} \right) + \lambda^2 \rho_k^3 = 0, \tag{78}$$

and then computing

$$h_{k+1} = \frac{2\rho_k}{\alpha\beta\overline{L}\|A\|}. \tag{79}$$

Because of the complexity of (78), we must solve for ρ_k numerically.

4 Examples

In this section we explore this variable step-size selection method for two problems, the Lotka-Volterra model and the Kepler problem.

4.1 The Lotka-Volterra Model

For this example we consider the Lotka-Volterra model of a simple predator-prey system from mathematical biology. This particular example is taken from Hairer, Lubich, and Wanner [1]. Consider the following system:

$$\begin{bmatrix} \\ \dot{u} \\ \\ \dot{v} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ u(v-2) \\ \\ v(1-u) \\ \\ \end{bmatrix} = f(u,v); \quad t \in [0,50]. \tag{80}$$

In (80), $u(t)$ denotes the number of predators present at time t, $v(t)$ represents the number of prey present at time t, and the constants one and two have been chosen arbitrarily. This system was integrated numerically using the implicit midpoint rule. Since the system is non-linear and the Lipschitz constant of the system as a whole is unknown, we will use the method described in Section 3.3.

This procedure was compared to a fixed step-size integration method with random step-sizes chosen. Two measures were chosen for comparison. The first measure, T, was total cpu time (in seconds) for 1000 runs with random initial data uniformly distributed on $[0.1, 10]$. The second measure, E, was the maximum absolute variation of the numerical method from

$$I(u,v) = \ln u - u + 2\ln v - v, \tag{81}$$

an invariant quantity for this system. The initial data for the system in this case was chosen to be $[u(0)\ \ v(0)]^T = [2\ \ 6]^T$.

We found that for simple systems such as (80), the numerical computational overhead in computing the step-size in the optimal h method renders the method less useful than a simple fixed step-size method. After trying various fixed step-sizes, it was determined that for 1000 runs with random initial data, $h = 0.125$ was the largest fixed step-size attempted that permitted convergence. For $h = 0.125$, $T = 118.258$ and $E = 0.064$. For the optimal h method, various values for λ were tried until a comparable value for E was found. For instance, for $\lambda = 2$ we get $E = 0.143$; for $\lambda = 3$ we get $E = 0.068$; and for $\lambda = 4$ we get $E = 0.037$. Since $\lambda = 3$ yielded a comparable value of E, $\lambda = 3$ was chosen for 1000 runs with random initial data and it was found that $T = 195.570$.

Different results arise when we try more challenging problems. Consider this variation to the Lotka-Volterra problem:

$$\begin{bmatrix} \dot{u} \\ \\ \dot{v} \end{bmatrix} = \begin{bmatrix} u^2 v(v-2) \\ \\ v^2 u(1-u) \end{bmatrix} = f(u,v); \quad t \in [0, 50]. \tag{82}$$

This system has has the same invariant as (80), but is very sensitive to random initial data. For this reason the initial data is fixed at $[u(0)\ \ v(0)]^T = [2\ \ 3]^T$ for the computation of both T and E.

Two methods were chosen to solve for the stage value y_1 which is defined implicitly by (10). The first method is the algorithm of Section 2, which is simply a Picard iteration. Secondly, we used Newton's method to compute the stage value y_1. The results from this example are given in Tables 4.1.1

through 4.1.4 and Figures 4.1.2 and 4.1.3.

To compare the fixed step-size method to the variable step-size method, we must locate times that are comparable from the tables and then compare the equivalent error. For example, we first notice that for the fixed step-size $h = 0.1$ in Table 4.1.1, the method took 160.701 seconds to solve the problem using a Picard iteration to solve for y_1. The error involved for this step-size was 0.094. Now we look in Table 4.1.2 and find that when $\lambda = 2$, the problem was solved in 168.412 seconds, which is about eight seconds longer than for $h = 0.1$. However, we notice that the error has been reduced to 0.004, which is about 4% of the error from when $h = 0.1$. We can locate other instances similar to this from the two tables. For the fixed step-size $h = 0.08$, the problem is solved in 234.427 seconds using Newton's method to find y_1, yielding an error of 0.069. We compare this to $\lambda = 2$ which was solved in 229.851 seconds with an error of 0.004. In addition, for the fixed step-size $h = 0.125$ using Newton's method to solve for y_1, the problem is solved in 155.564 seconds with an error of 0.084. We compare this to $\lambda = 1$ in which the problem is solved in 151.649 seconds with an error of 0.025.

In Table 4.1.2, when we computed the stage value y_1 using Newton's method, we actually used the variable step-sizes determined by the solution of (52) not (78). For comparison, we recreated these table on another machine; this time we use (78) to compute the variable step-sizes when we are solving for the stage value y_1 using Newton's method. The results are given in Table 4.1.3 and Table 4.1.4. There are a couple of interesting things to point out. First, we notice that for fixed step-sizes, Newton's method works faster than the fixed-point iteration. We must note that the MATLAB® code that generated the data for Tables 4.1.1 through 4.1.4 was exactly the same; the only difference is

that Tables 4.1.3 and 4.1.4 contain data from a different version of MATLAB® and ran on a different machine. The main goal of this paper is not to compare Newton's method versus Picard iteration as much as it is to compare fixed step-sizes to variable step-sizes. In this regard, the results shown in Tables 4.1.3 and 4.1.4 agree with those given in Tables 4.1.1 and 4.1.2. For example, when we use the fixed step-size $h = 0.05$, the problem is solved in 67.701 seconds with an error of 0.031 when solving for the stage value using a Picard iteration. Using the variable step-size method with $\lambda_P = 3.6$, we see that the problem is solved in 67.413 seconds with an error of 0.003, which is about 90% smaller than the fixed-step size error of 0.031. A similar comparison can be made with $h = 0.125$ and $\lambda_P = 2$.

The second interesting thing we notice is evidenced by Table 4.1.4 when we use (78) to compute the variable step-sizes and solve for the stage value y_1 using Newton's method. We notice that the problem takes longer to solve, but at the same time the error is much smaller. We cannot compare the fixed step-size data with the variable step-size data as above because of the large difference in times. However we can note that when the problem is solved with $\lambda_P = 0.4$ the problem is solved in 27.097 seconds with an error of 0.087; when the problem is solved with $\lambda_N = 10$, it takes 115.689 seconds to get the error down to 0.001130. We can quickly compute that the problem is solved 77% faster using λ_P, but the error is 98% lower using λ_N.

The reason it takes so long to solve the problem using this method is because of the variable step-sizes determined by solving (78). Consider the plot shown in Figure 4.1.4. This graph shows how far the integration of the problem has progressed at each iteration for various values of λ_P and λ_N. Essentially, the graph also shows how large (or how small) of a step-size each method chooses

as the integration proceeds from one step to the next. In addition, we can see from the plot that when we use (78) to find the step-size update, the step-sizes determined are much smaller then when (52) is used; hence, many more iterations are required to integrate the system from $t = 0$ to $t = 50$. We would also like to mention something about the existence of a solution to (78). Although, we have not proven that a solution does exist, we can plot the function for reasonable values of the parameters. For example, consider Figure 4.1.5. Here we plot the function $g(\rho)$ where g is the function given by the left hand side of (78). This plot uses the following values for the parameters given in (78): $\lambda = 10$, $\varepsilon = 10^{-10}$ and $\alpha = \left\| [2 \;\; 3]^T \right\|$. The function appears to be quite smooth in this region and clearly we see that for these parameters, a solution to (78) exists.

As one can see from the example above, inherent with this variable step-size selection method is the choice of the parameter λ. We will use the system given by equation (82) to explain how one should choose an appropriate value of λ when integrating a system that evolves over a long period of time. Suppose we are interested in integrating the system described by (82) over the interval $[0, 500]$ or $[0, 1000]$. First, we choose a much smaller value for the final time of integration; in this example that value is $T = 50$. We then integrate the system over the interval $[0, 50]$ with a fixed step-size and at the same time with various values of λ. Essentially, we analyze how λ affects this system in particular, just as we did in the above example. After we have integrated the system over the much smaller time interval, we choose the value of λ that works best for this system to integrate the system over the entire time interval. This process should be done for any system where the length of the interval over which the integration must be performed is quite large when compared

to the evolution of the dynamics of the system.

All computations, except those that generated Tables 4.1.3 and 4.1.4, were done in MATLAB® version 6.1.0.450 Release 12.1 running in Microsoft Windows XP Professional version 5.1.2600 with an AuthenticAMD processor running at 1544 Mhz.

4.2 The Kepler Problem

This example, taken from Hairer, Lubich, and Wanner [1], is the well known example describing planetary motion discovered by J. Kepler. We shall consider the following two-body problem:

$$\ddot{q}_1 = -\frac{q_1}{(q_1^2 + q_2^2)^{3/2}} \tag{83}$$

$$\ddot{q}_2 = -\frac{q_2}{(q_1^2 + q_2^2)^{3/2}}, \tag{84}$$

with the following initial conditions:

$$q_1(0) = 1 - e, \quad q_2(0) = 0, \quad \dot{q}_1(0) = 0, \quad \dot{q}_2(0) = \sqrt{\frac{1+e}{1-e}}, \tag{85}$$

where $0 \leq e < 1$. To describe the motion of two bodies, one of the bodies is taken to be the center of the coordinate system and the position of the second at any time t is given by the two coordinates $q_1(t)$ and $q_2(t)$. Equations (83)-(84) are equivalent to the following Hamiltonian system:

$$\dot{q}_i = p_i \quad i = 1, 2 \tag{86}$$

$$H(p_1, p_2, q_1, q_2) = \frac{(p_1^2 + p_2^2)}{2} - \frac{1}{\sqrt{q_1^2 + q_2^2}} \tag{87}$$

where $H(p_1, p_2, q_1, q_2)$ is the Hamiltonian of the system.

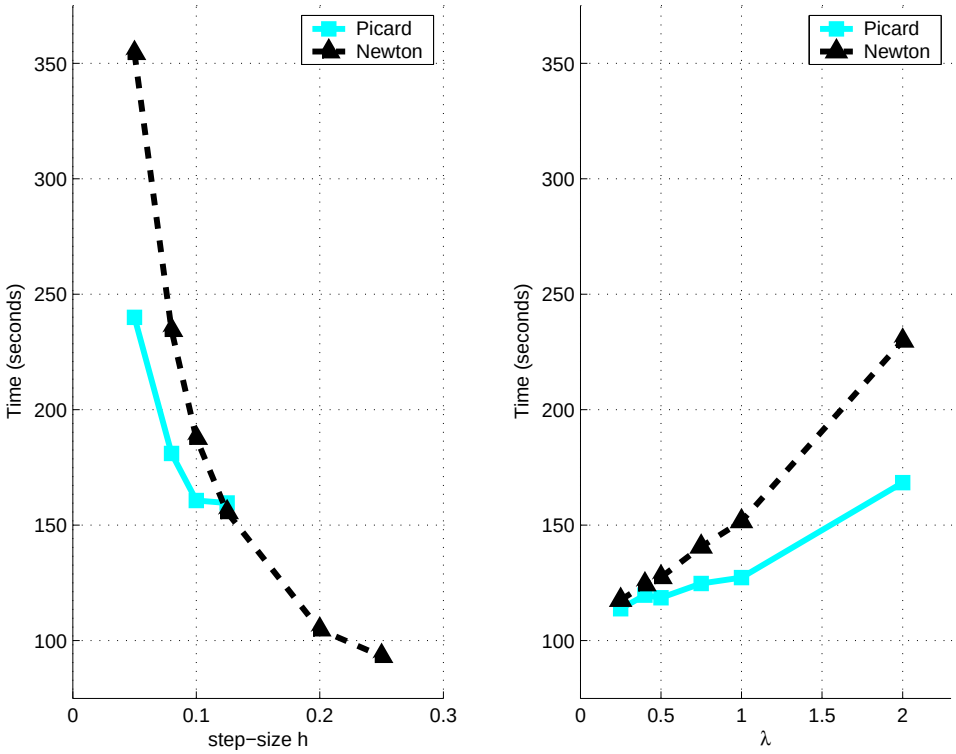

Fig. 4.1.2. Comparison of execution time for h and λ for the Lotka-Volterra model

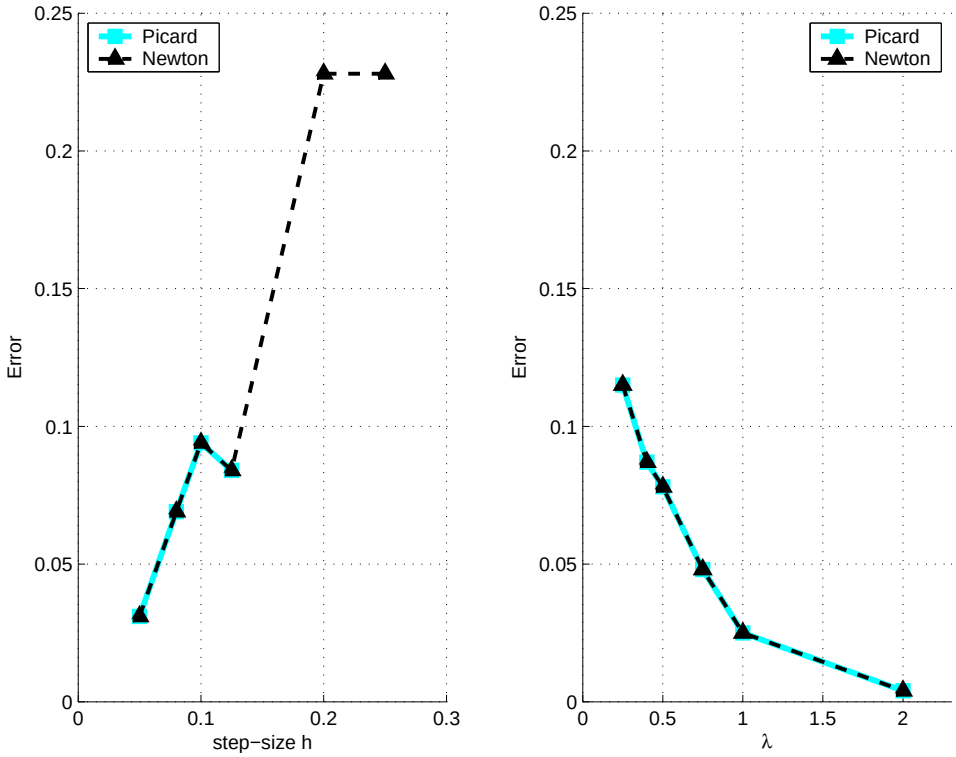

Fig. 4.1.3. Error comparison for h and λ for the Lotka-Volterra model

Before (83)-(84) can be integrated using the implicit midpoint rule, we convert the equations to a system of four first-order ordinary differential equations. Let $z_1 = q_1,\ z_2 = \dot{q}_1,\ z_3 = q_2$ and $z_4 = \dot{q}_2$. Define $z = [z_1\ z_2\ z_3\ z_4]^T$. Then (83)-(84) are equivalent to the following system:

$$\dot{z} = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} z_2 \\ -\frac{z_1}{\left(z_1^2+z_3^2\right)^{3/2}} \\ z_4 \\ -\frac{z_3}{\left(z_1^2+z_3^2\right)^{3/2}} \end{bmatrix}. \tag{88}$$

The above system of equations was solved using the implicit midpoint rule for $t \in [0, 50]$. Just as in the previous example, both a Picard iteration and Newton's method were used to compute the stage value y_1. The two measures we chose for this example are very similar to the those of the previous example. The first measure is T, the total cpu time required to solve the system 1000 times with eccentricity, e, that is uniformly distributed in the interval [0.4,0.8]. The second measure was E, the maximum absolute error that the integration deviates from the exact solution with eccentricity $e = 0.6$. For this measure, we considered the absolute error at every step of the integration.

The computations were performed on the same machine as the previous example, and the results are summarized in Tables 4.2.1 and 4.2.2 and also in Figures 4.2.1 and 4.2.2. We compare the performance of the variable step-size method to the fixed-step size method exactly as we did in the previous example. In Table 4.2.1 we begin with the fixed step-size $h = 0.01$. The problem is solved in 84.842 seconds using Picard iteration to find y_1, giving an error of 0.113. We compare this to $\lambda = 8$ in Table 4.2.2 where the problem is solved

in 81.388 seconds with an error of 0.067. Also, when the fixed step-size is $h = 0.05$, the problem is solved in 21.190 seconds using Picard iteration to find y_1 and is solved in 29.602 using Newton's method to find y_1. The error in both cases is 1.581. We compare these to when $\lambda = 2$ in Table 4.2.2. Respectively, the times are 21.572 seconds and 29.943 seconds. The error for these two times is 0.643.

4.3 The Kepler Problem Using Runge-Kutta-Nyström

Once again, we consider the Kepler problem described by equations (83)-(85). We now solve the problem using a Runge-Kutta-Nyström (RKN) method. The method we use is the two-stage Gauss method from [3] which is the RKN method that is induced from the fourth order Gauss method (a Runge-Kutta routine) found in [1]. For notation, we will refer to this integration routine as Gauss4. The constants from equations (28)-(30) are

$$\gamma_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}, \;\; \gamma_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}, \;\; \beta_1 = \frac{1}{4} + \frac{\sqrt{3}}{12}, \;\; \beta_2 = \frac{1}{4} - \frac{\sqrt{3}}{12}, \;\; b_1 = \frac{1}{2}, \quad (89)$$

$$b_2 = \frac{1}{2}, \;\; a_{11} = \frac{1}{24}, \;\; a_{12} = \frac{1}{8} - \frac{\sqrt{3}}{12}, \;\; a_{21} = \frac{1}{8} + \frac{\sqrt{3}}{12}, \;\; a_{22} = \frac{1}{24}. \quad (90)$$

The analysis leading to equation (51) used the Lipschitz constant from the Runge-Kutta methods, $h\overline{L}\|A\|$. Since we are actually implementing a RKN method to solve this system now, we have to repeat that analysis process with the Lipschitz constant from the RKN methods, $h^2\overline{L}\|A\|$. Following exactly the same process given in equations (34)-(52) and using the substitution

$$x = h^2\overline{L}\|A\|, \quad (91)$$

we find that to find the optimal h we must solve

$$2 + \ln x + \lambda^2 x^{3/2} = 0 \tag{92}$$

for x and then solve for h using (91).

Tan [5] solves the Kepler problem using Gauss4 as a Runge-Kutta routine (not in its induced RKN form) with a fixed step-size of $\frac{\pi}{64}$ for 500,000 iterations. Just for comparison we did the following:

(1) We solve the problem using the fixed step-size $\frac{\pi}{64}$ on the interval [0,50] and determine the maximum absolute variance from the exact solution.
(2) We compute the total cpu time taken to solve the system using this step-size for 500,000 iterations as Tan does.
(3) We solve the system on the interval [0,50] using the optimal step-size selection method to determine the value of λ that gives us a comparable error as determined in step one.
(4) We solve the system on the entire interval using this value of λ and our variable step-size selection method.

When we used $h = \frac{\pi}{64}$ to solve the system on the interval [0,50], we found the maximum absolute variance from the exact solution to be 0.0045. Then using this fixed step-size, we found that it took 101.577 cpu seconds to solve the problem taking 500,000 iterations. We then determined that a comparable error on the interval [0,50] was achieved for $\lambda = 55$; that error was 0.0046. When we solve the problem using our variable step-size selection method and choose $\lambda = 55$ to a final time of $T = \frac{500000\pi}{64}$, we find that the system is solved in 61.461 cpu seconds, which is 39.5% faster than using the fixed step-size of $\frac{\pi}{64}$.

In addition to the above comparison, we also compared the fixed step-size method to the variable step-size method for various values of h and λ for this example. The results of this comparison are given in Tables 4.3.1 and 4.3.2. We only used Picard iteration to solve for the stage values. We used exactly the same two measures as we did in the previous section, when we solved the Kepler problem using the implicit midpoint rule. When we solve the system with the fixed step-size $h = 0.05$, the solution is found in 198.247 seconds with an error of 0.0050. Comparatively, we found that for $\lambda = 100$, the system is solved in 193.291 seconds with an error of 0.0036, which is about 28% lower error. We also found that when the system is solved with the fixed step-size of $h = 0.06$, it took 179.498 seconds with an error of 0.0071. We compare this to the variable step-size with $\lambda = 90$, which took only 182.738 seconds to once again get an error of 0.0036, which is about 49% lower error. In addition to these positive results, we did not notice a large decrease in the error using the variable step-size method as we increased λ. When we used the implicit midpoint rule in the previous two examples, we noticed a steady decline in the error as λ increased. In this case, we actually noticed that the error only goes from 0.0036 when $\lambda = 90$ to 0.0032 when $\lambda = 1000$. We point out that when $\lambda = 1000$, the system is solved in 625.772 seconds which is about 33% faster than the 927.805 seconds it took with the fixed step-size $h = 0.01$, where the error was only 0.0029.

All computations were done in MATLAB® version 6.5.0.180913a Release 13 running in Microsoft Windows XP Professional version 5.1.2600 with an x86 Family 15 Model 2 Stepping 7 GenuineIntel processor running at 2392 Mhz.

5 Conclusion

The collection of implicit numerical integration routines is vast to say the least. Often times one routine is chosen over another to improve either efficiency or accuracy. In this paper, we have shown that it is possible to wisely choose a variable step-size for these integration schemes.

For linear ordinary differential equations or equations in which the Lipschitz constant for the function f is known, the task becomes quite simple as the optimal value of the step-size will not change from one step of the integration to the next. But, if we are dealing with more complicated non-linear differential equations, we can still choose an optimal time step at each step of integration of the system. As we have shown, this process often involves solving a non-linear equation numerically. Because of this, the computational overhead in using this optimal step-size routine seems to be too much for solving differential equations in which the function f is quite simple. However, our results have shown that this is consistently not the case when f is a complicated function as we describe below.

Tables 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.2.1 and 4.2.2 clearly show that, for comparable integration times, the variable step-size selection method presented in this paper drastically reduces the global error in the solution of the problem. For the Kepler problem, we found that for comparable execution times, the error was reduced 41% to 59% when the variable step-size method is used. In the Lotka-Volterra example, we found that for comparable execution times, the problem is solved with the error reduced 70% to 96% when we use the variable step-size method. From studying the tables we may choose λ so that

execution times are comparable, in which case the variable step-size method noticeably reduces error as evidenced from the above discussion. However, λ may also be adjusted to find comparable errors between the fixed step-size and variable step-size methods. When you do this, one notices that the time required to achieve a comparable error for the fixed step-size is much larger.

We must point out that this optimal step-size selection process is dependent upon the scheme being used and we have concentrated on Runge-Kutta and Runge-Kutta-Nyström methods. It should not be too difficult of a task to adapt this process to the ever growing collection of implicit integration routines.

References

[1] E. Hairer, C. Lubich, G. Wanner, Geometric Numerical Integration - Structure-Preserving Algorithms for Ordinary Differential Equations, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, Germany, 2002.

[2] D. Lewis, J. Simo, Conserving algorithms for the N-dimensional rigid body, Fields Institute Communications 10 (1996) 121–139.

[3] J. Sanz-Serna, M. Calvo, Numerical Hamiltonian Problems, Applied Mathematics and Mathematical Computation, Chapman & Hall, London, United Kingdom, 1994.

[4] L. W. Roeger, A class of nonstandard symplectic discretization methods for Lotka-Volterra system, preprint - Texas Tech University Department of Mathematics and Statistics (2005).

[5] X. Tan, Almost symplectic Runge-Kutta schemes for Hamiltonian systems, Journal of Computational Physics 203 (1) (2005) 250–273.

[6] E. Hairer, G. Wanner, Algebraically stable and implementable Runge-Kutta methods of high order, SIAM Journal on Numerical Analysis 18, 1981.

[7] E. Hairer, G. Wanner, Characterization of non-linearly stable implicit Runge-Kutta methods, in: J. Hinze (Ed.), Numerical Integration of Differential Equations and Large Linear Systems, Vol. 968 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, Germany, 1982, pp. 207–219, proceedings of Two Workshops Held at the Univeristy of Bielefeld, Spring 1980.

[8] T. Hull, W. Enright, B. Fellen, A. Sedgwick, Comparing numerical methods for ordinary differential equations, SIAM Journal on Numerical Analysis 9 (4) (1972) 603–637.

[9] R. Bulirsch, J. Stoer, Numerical treatment of ordinary differential equations by extrapolation methods, Numerische Mathematik 8 (1966) 1–13.

[10] J. Stoer, R. Bulirsch, Introduction to Numerical Analysis, Third Edition, Texts in Applied Mathematics 12, Springer-Verlag, New York, New York, 2002.

[11] P. V. D. Houwen, Construction Of Integration Formulas For Initial Value Problems, Vol. 19 of North-Holland Series In Applied Mathematics And Mechanics, North-Holland Publishing Company, Amsterdam, 1977.

[12] B. Cano, A. Duran, Analysis of variable-stepsize linear multistep methods with special emphasis on symmetric ones, Mathematics of Computation 72 (244) (2003) 1769–1801.

[13] A. Stuart, A. Humphries, Dynamical Systems and Numerical Analysis, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, Cambridge, United Kingdom, 1998.

[14] C. V. Loan, The ubiquitous Kronecker product, Journal of Computational and Applied Mathematics 123 (2000) 85–100.

[15] D. Kincaid, W. Cheney, Numerical Analysis, Second Edition, Brooks/Cole Publishing Company, Pacific Grove, CA, 1996.

[16] L. Collatz, Functional Analysis and Numerical Analysis, Academic Press, New York, 1966.

[17] J. Ortega, W. Rheinboldt, Iterative solution of non-linear equations in several variables, Academic Press, New York, 1970.

Table 4.1.1

Fixed step-size (Lotka-Volterra model)

$h \rightarrow$	0.05	0.08	0.1	0.125	0.2	0.25
T (Picard)	239.995	181.081	160.701	159.590	DNC	DNC
E (Picard)	0.031	0.069	0.094	0.084	DNC	DNC
T (Newton)	354.380	234.427	187.720	155.564	104.811	93.294
E (Newton)	0.031	0.069	0.094	0.084	0.228	0.228

Table 4.1.2

Variable step-size (Lotka-Volterra model)

$\lambda \rightarrow$	0.25	0.4	0.5	0.75	1	2
T (Picard)	113.834	119.692	118.531	124.759	127.323	168.412
E (Picard)	0.115	0.087	0.078	0.048	0.025	0.004
T (Newton)	117.586	124.294	127.428	140.652	151.649	229.851
E (Newton)	0.115	0.087	0.078	0.048	0.025	0.004

Table 4.1.3

Fixed step-size (Lotka-Volterra model)

$h \rightarrow$	0.05	0.065	0.08	0.1	0.125
T (Picard)	67.701	57.018	50.742	46.178	45.596
E (Picard)	0.031	0.050	0.069	0.094	0.084
T (Newton)	67.801	53.260	43.990	35.574	29.636
E (Newton)	0.031	0.050	0.069	0.094	0.084

Table 4.1.4

Variable step-size (Lotka-Volterra model)

$\lambda_P \rightarrow$	0.4	0.6	0.8	1	2	3.6
T (Picard)	27.097	28.377	28.761	33.902	43.595	67.413
E (Picard)	0.087	0.066	0.044	0.025	0.004	0.003
$\lambda_N \rightarrow$	8	10	12	14	16	20
T (Newton)	192.345	115.689	138.058	134.097	140.481	149.867
E (Newton)	0.000459	0.001130	0.000841	0.000891	0.000819	0.000734

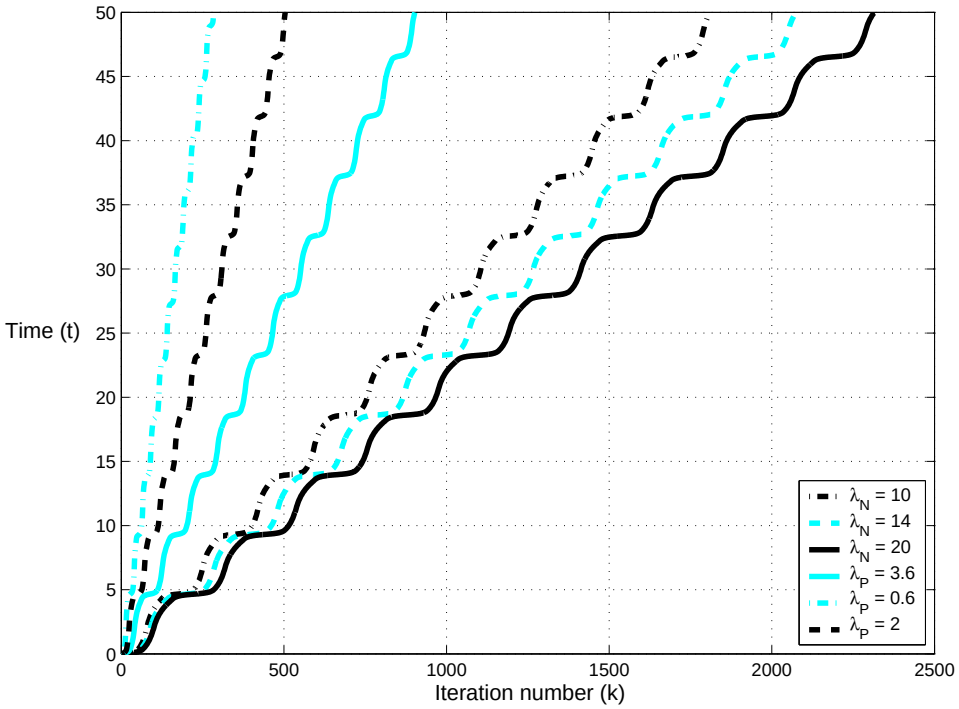

Fig. 4.1.4. Comparison of the variable step-sizes determined by solving (52) versus (78)

Table 4.2.1

Fixed step-size (Kepler problem)

$h \rightarrow$	0.01	0.05	0.1	0.125	0.2	0.25
T (Picard)	84.842	21.190	13.299	11.858	DNC	DNC
E (Picard)	0.113	1.581	2.038	2.483	DNC	DNC
T (Newton)	137.506	29.602	16.503	14.391	DNC	DNC
E (Newton)	0.113	1.581	2.038	2.483	DNC	DNC

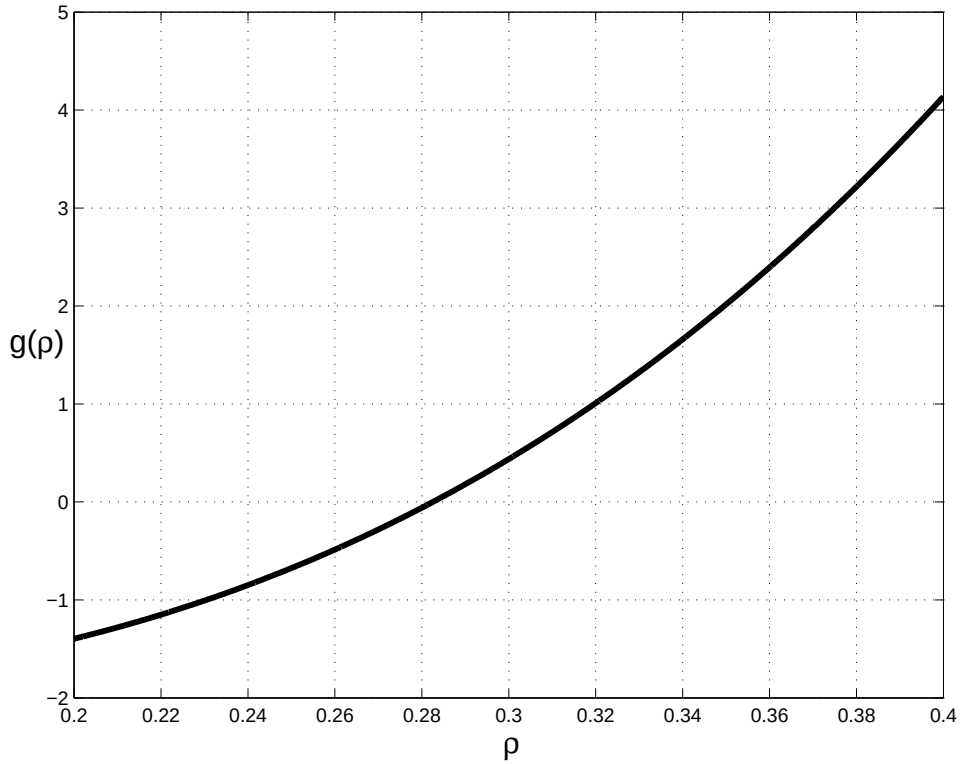

Fig. 4.1.5. Plot of ρ versus $g(\rho)$, where g is given by the left hand side of (78)

Table 4.2.2

Variable step-size (Kepler problem)

$\lambda \rightarrow$	1	2	4	6	8	10
T (Picard)	20.199	21.572	37.234	58.505	81.388	111.440
E (Picard)	1.250	0.643	0.295	0.133	0.067	0.037
T (Newton)	22.062	29.943	50.602	79.263	114.673	157.194
E (Newton)	1.250	0.643	0.295	0.133	0.067	0.037

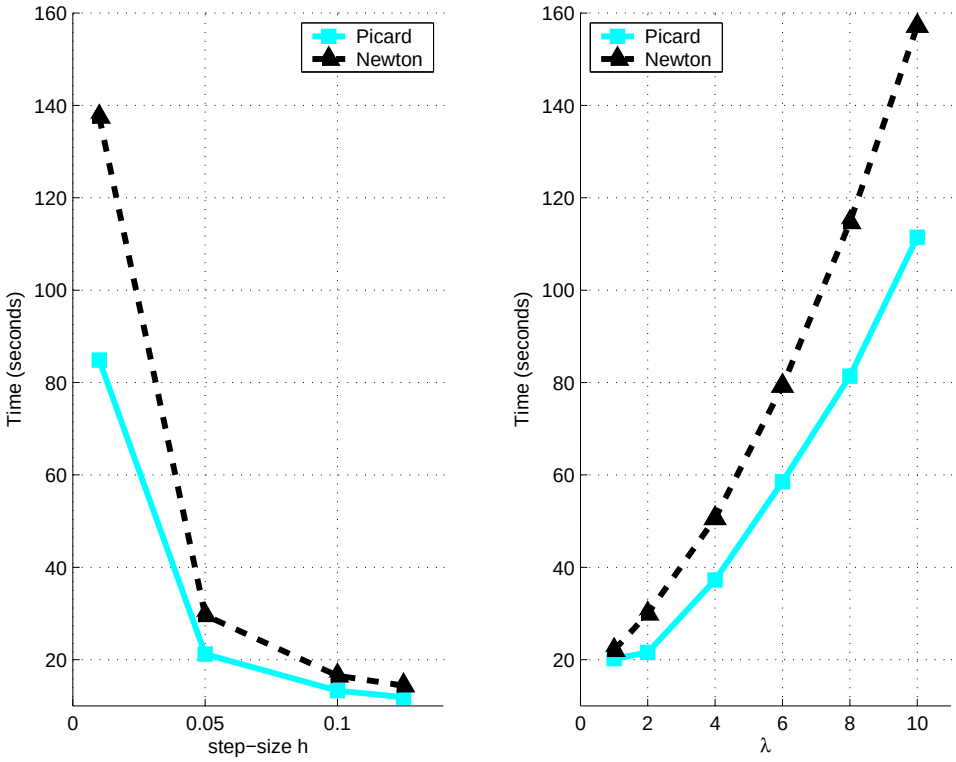

Fig. 4.2.1. Comparison of execution time for h and λ for the Kepler problem

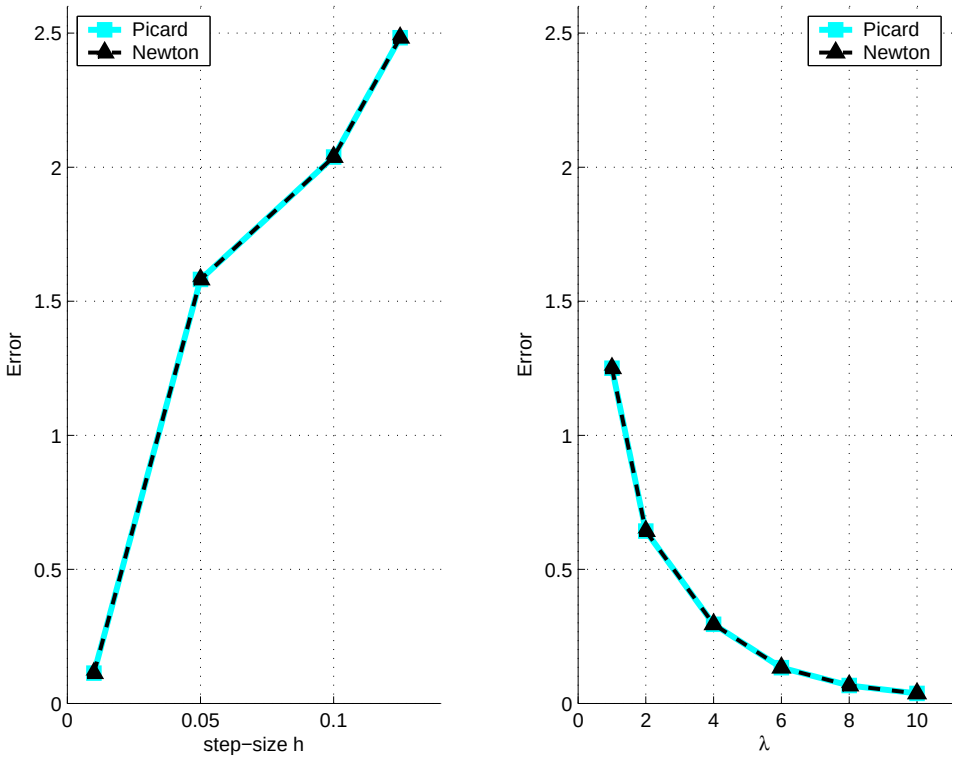

Fig. 4.2.2. Error comparison for h and λ for the Kepler problem

Table 4.3.1

Fixed step-size (Kepler problem using RKN)

$h \rightarrow$	0.01	0.025	0.05	0.06	0.1
T (Picard)	927.805	341.167	198.247	179.498	DNC
E (Picard)	0.0029	0.0030	0.0050	0.0071	0.0508

Table 4.3.2

Variable step-size (Kepler problem using RKN)

$\lambda \rightarrow$	90	100	250	500	1000
T (Picard)	182.738	193.291	248.634	352.381	625.772
E (Picard)	0.0036	0.0036	0.0032	0.0032	0.0032